Hindawi
Scientific Programming
Volume 2017, Article ID 2573592, 11 pages
https://doi.org/10.1155/2017/2573592

Research Article

An Association-Oriented Partitioning Approach for Streaming Graph Query

Yun Hao,[1,2,3] Gaofeng Li,[1,2,3] Pingpeng Yuan,[1,2,3] Hai Jin,[1,2,3] and Xiaofeng Ding[1,2,3]

[1]*Services Computing Technology and System Lab., School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China*
[2]*Cluster and Grid Computing Lab., School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China*
[3]*Big Data Technology and System Lab., School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China*

Correspondence should be addressed to Pingpeng Yuan; ppyuan@mail.hust.edu.cn and Hai Jin; hjin@hust.edu.cn

Received 26 December 2016; Accepted 11 April 2017; Published 25 May 2017

Academic Editor: Alex M. Kuo

The volumes of real-world graphs like knowledge graph are increasing rapidly, which makes streaming graph processing a hot research area. Processing graphs in streaming setting poses significant challenges from different perspectives, among which graph partitioning method plays a key role. Regarding graph query, a well-designed partitioning method is essential for achieving better performance. Existing offline graph partitioning methods often require full knowledge of the graph, which is not possible during streaming graph processing. In order to handle this problem, we propose an association-oriented streaming graph partitioning method named Assc. This approach first computes the rank values of vertices with a hybrid approximate PageRank algorithm. After splitting these vertices with an adapted variant affinity propagation algorithm, the process order on vertices in the sliding window can be determined. Finally, according to the *level* of these vertices and their association, the partition where the vertices should be distributed is decided. We compare its performance with a set of streaming graph partition methods and METIS, a widely adopted offline approach. The results show that our solution can partition graphs with hundreds of millions of vertices in streaming setting on a large collection of graph datasets and our approach outperforms other graph partitioning methods.

1. Introduction

With the rapid development of Internet, huge amounts of graph data are emerged every day. For example, the Linked Open Data Project which aims to connect data across the web published 149 billion triples till 2017 while 31 billion triples were published in September 2011 [1]. In addition, dynamic real-world graphs include social networks. The dynamicity of graphs stems from either the world-wide hot events or updates of the web contents [2]. Thus, the rapid explosion of data volume and dynamicity urgently necessitates large scale graph analysis applications which can handle these dynamic workloads. To achieve a desirable performance in big dynamic graph analysis, streaming graph partitioning algorithms, which give a guarantee of scalability by distributing a streaming graph to multiple machines, will be employed inevitably.

In fact, graph partitioning problem has received a lot of attention over the last years. The existing algorithms may be grouped in two divisions: edge cut algorithms and vertex-cut algorithms. The majority of distributed graph engines adopt edge-based hash partitioning [3–6] as the data partitioning solution. Edge-based hash partitioning is a vertex-cut approach which distributes edges across the partitions by computing the hash keys of vertices and allows edges to span partitions. For example, Pregel [7] uses a special hash function to distribute the vertex. The principle of this approach is to collect those edges which share the same vertex and distribute the vertices to different computing nodes. Hence, this approach can obtain good performance for simple graph operations, such as answering star queries. Although hash approach generates a balanced number of vertices across distributed computing nodes, it entirely ignores the graph structure. As a result, many messages have to be sent across

the nodes when executing graph operations. It leads to heavy communication traffic. All in all, this approach has extremely poor locality, which means that the complex graph query operations may incur frequent cross-node interactions and intermediate result exchanging. Consequently, the benefit of distributed and parallel processing of the big graph data is lost or significantly reduced due to the high communication overhead incurred by hash partitioning scheme. Wu et al. proposed path partitioning for scalable SPARQL query processing over static RDF graphs [8]. Path partitioning approach does not divide a graph into a set of independent edges or vertices but sets of paths. Thus, the approach can largely reduce the cost of distributed joins over the large RDF dataset.

In this paper, we propose an association-oriented partitioning approach for streaming graphs. Our approach is based on one important observation: in order to minimize the interactions among partitions, we need to consider the associations among vertices when we assign vertices and edges to partitions. The main contributions of the paper are twofold. *First*, for a streaming situation in which large scale graph data arrives fast and continuously, we propose an association-oriented partitioning approach. The approach considers the association among recent arriving graph data which falls in a sliding window. The approach first computes rank scores of vertices in the sliding window and then clusters the vertices and edges according to the rank values of vertices and associations between vertices. By partitioning the big graphs into multiple partitions with this approach, we reduce interactions among partitions and the cost for internode communication. *Second*, we evaluate our approach in labeled and unlabeled streaming graphs through extensive experiments. The experimental results show that our approach outperforms HASH approach and METIS [9]. The reason is that our approach reduces the interactions between partitions. The results also show that our approach is capable of handling incrementally generated streaming data.

2. Preliminary

In this section, we will present the concepts used in the paper. Our approach can handle both directed and undirected graphs. In addition, labeled and unlabeled graphs can be processed, too. Since undirected graph can be easily transformed into directed graph by adding another edge between two connected vertices, the following discussion mainly focuses on directed connected graph defined in [5, 6].

Generally, the edges or vertices, or both, of a graph are assigned labels. A graph with labeled vertices is named as a vertex-labeled graph. Similarly, in an edge-labeled graph each edge has a label. In a directed edge-labeled graph, the label of an edge indicates the relationship between its source vertex and target vertex. An edge with its two vertices ($a \xrightarrow{b} c$) can be represented as a triple (a, b, c). In RDF (Resource Description Framework) graph (e.g., Figure 1(b), a fragment from LUBM [10]), a is called subject and c is the object of the triple. Label b on the edge is the predicate of the triple. In the following, labeled graph will be defined formally.

Definition 1 (association of vertices). Regarding a graph $G(V, E)$, the association between handling u and v is defined as $A(u, v)$; min(mvn) is the minimal number of vertices on a path that connects the two vertices:

$$A(u,v) = \begin{cases} 1 & \text{directly connected} \\ \dfrac{1}{\min(\text{mvn}) + 1} & \text{indirectly connected} \\ 0 & \text{unconnected.} \end{cases} \quad (1)$$

A graph partitioner assigns vertices or edges to k partitions. The associations of the partitions should be low in order to reduce the interactions among partitions. Here we define the association of two clusters.

Definition 2 (association of clusters). Assume two clusters (partitions) C_a and C_b. e_a, e_b are the exemplars of clusters C_a, C_b. Exemplar is found by randomly choosing an initial subset of data points and then iteratively refining it. The association $A(C_a, C_b)$ between cluster C_a and cluster C_b is defined as follows:

$$A(C_a, C_b) = \begin{cases} A(e_a, e_b) & e_a, e_b \text{ are connected} \\ 0 & \text{otherwise.} \end{cases} \quad (2)$$

Graph partitioning approach should distribute vertices to each cluster uniformly. It is also critical that the total number of cross-partition edges is small, in order to minimize the communication cost between different partitions.

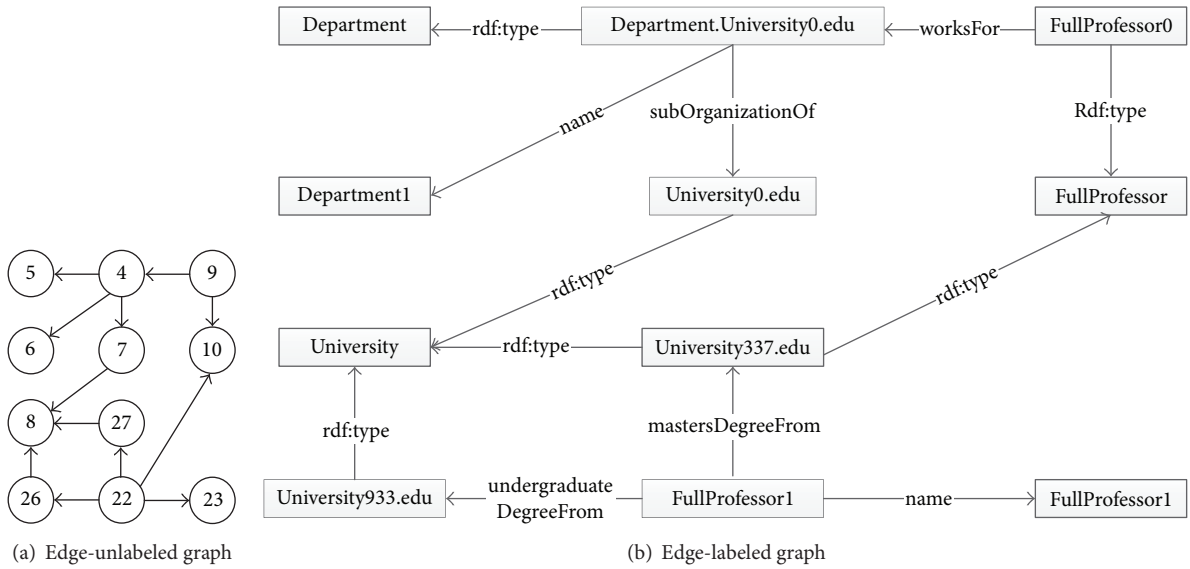
Definition 3 (edge cut). Assume graph $G(V, E)$, where V is the set of vertexes in the graph and E is the set of edges in the graph. Assume a partition of G consists of $P_1, P_2, \ldots, P_n$. Namely, $P_1 \cup P_2 \cup \cdots \cup P_n = G$. For partition P_i, $\forall v \in P_i$,

$$\text{ec}(v) = \begin{cases} 1 & ((v,t) \in G \vee (t,v) \in G) \wedge t \notin P_i \\ 0 & \text{otherwise;} \end{cases} \quad (3)$$

the edge cut of P_i is $\text{ec}(P_i) = \sum_{v \in P_i} \text{ec}(v)$. The edge cut of the partition is $\sum_{0 \le i < n} \text{ec}(P_i)$.

Minimizing the number of *edge cuts* may not be the only goal for partitioning approaches, as the cost of processing graphs is determined by not only the network communication but also the size of the messages, although each message typically is small and contains a little information. So it is not enough to minimize the total number of edges crossing different partitions. Minimizing the *communication volume* across computing nodes is also a goal.

Definition 4 (communication volume). Assume $P_1, P_2, \ldots, P_n$ is a partition of graph $G(V, E)$. $P_1 \cup P_2 \cup \cdots \cup P_n = G$. Communication volume $\text{cv}(P)$ is positive correlation with edge cut.



(a) Edge-unlabeled graph (b) Edge-labeled graph

Figure 1: Example of unlabeled and labeled graph.

3. Association-Oriented Streaming Graph Partitioning

In this section, we present our association-oriented approach for streaming graph partitioning. Streaming graph partitioning tries to distribute nodes and edges into multiple machines, while it should keep the data balance and the communication volume minimal. The streaming partitioner receives graph data and then decides where the nodes of the graph data should be distributed. Our approach first orders each vertex in partitions and then computes the association between a newly arrived vertex and other vertices which are distributed in a partition. Then we follow the rank values of vertices to merge the newly arrived vertices with other vertices. The rank values of vertices indicate the step in which vertices will be merged with others. At last, we use a variant of affinity propagation to cluster nodes. Thus, it faces two challenges. The first challenge is how to order the vertex that needs to be merged in the merging steps, as it may lead to extremely skewed data distribution if the order is not good enough. At the same time, both space and time complexity of association partitioning are high when the number of merging steps is large. More merged vertex means fewer replicas and higher intra-association. However, it leads to a skew in data distribution since there may be some very large clusters, which are undesirable. Consequently, we present an approach to keep the data distribution balanced. These techniques are discussed in terms of the light-weight AP-based stream graph clustering and association partitioning algorithm in following. More importantly, the processor will use some optimization strategies such as using start vertices instead of all vertices in the clustering process.

3.1. Hybrid Approximate PageRank Computation. Since the association merging is to merge vertices in each step, we need to order vertices first. The reason is to enable the vertices with a higher score to be merged later, because merging these vertices first may result in very large clusters. PageRank score of vertices can be used to rank vertices. At the same time, we have to group similar vertices by their PageRank score in order to simplify the computation. More specifically, for vertex v, we use a term *Level* (L) to indicate the level of the start vertices of each cluster that has the maximum association with vertex v, denoted by $L(v) = n$, $n \in \mathbb{N}^+$. Generally, the level starts from 1 and the level of the root is 1. Here, since our approach merges vertices from down to top, the level of leaves which is the deepest is 1. Namely, the level of a node is defined by the depth of the tree minus the number of connections between the node and the root. Sets of vertices which have similar PageRank values may have the same L value n, which exactly will be merged at the step n. By using L as the criterion for merging clusters, the number of steps in processing association partitioning can be reduced substantially.

The vertex rank-based grouping approach is to order the vertices in ascending order by their PageRank score. The PageRank for a directed graph can be seen as a way of ranking the entire distribution of the degree of each vertex. We first calculate the PageRank score for each vertex.

An important point to note is that it is not possible to access all the information in a streaming graph, in particular when the data are arriving continuously; that is, the volume of data is increasing. On the other hand, since arrived data has been distributed to storage nodes, it is difficult to analyze all of the data due to its excessive volume as well as the cost for data transmission. It is desirable if we can estimate the PageRank of some selected nodes quickly.

To address this problem, we approximate the PageRank of each vertex from its in-degree [11]. Equation (4) shows that the average PageRank of nodes k is proportional to its in-degree:

$$\overline{p}(k) = \frac{d}{N} + \frac{1-d}{N}\frac{k_{\text{in}}}{\langle k_{\text{in}} \rangle}. \tag{4}$$

Table 1: Proportion of each predicate in LUBM.

Predicate	Type	Name	teacherOf	worksFor	memberOf	Advisor
Proportion	20.0402%	15.4818%	1.5641%	0.5209%	7.5539%	2.9666%

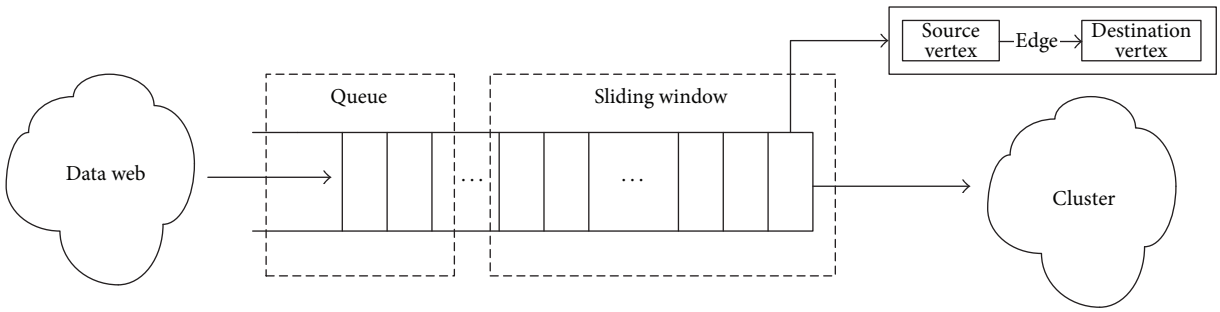


Figure 2: Sliding window.

Generally, PageRank algorithm does not consider the difference between edges. However, in labeled graphs, they may be different, which requires considering the edge weight. For example, in Figure 1(b), each triple in the RDF graph can be treated as a direct edge whose subject as a source points to object as a destination and the predicate as a label for the edge. The proportion of same predicates in triples is different. Experimental results show that the total number of predicates in LUBM [10, 12] is 18, and the selected six predicates in Table 1 show that each p_i has a relatively stable proportion to the total in a data set.

Suppose edge points from s_{sub} to o_{obj} with label p_i; the PageRank of o_{obj} contributed from s_{sub} is computed as

$$\text{PR}\left(s_{\text{sub}} \longrightarrow o_{\text{obj}}\right) = \frac{w_{p_i}}{\sum_{\forall s: s \xrightarrow{p} o_{\text{obj}}} w_p} \text{PR}\left(s_{\text{sub}}\right), \tag{5}$$

where w_{p_i} is the percentage of p_i in total number of p:

$$w_{p_i} = \frac{\left|p_i\right|}{\sum_{j<\|P\|} \left|p_j\right|}, \tag{6}$$

and the sum of w_{p_j} is the total percentage of p that s_{sub} has. The quotient means the weight of p_i assigned by the subject s_{sub}. Multiplication by PageRank of s_{sub} is the contribution that s_{sub} gives to o_{obj}.

Then a hybrid approximate algorithm for computing the PageRank score quickly in the sliding window is as follows:

$$\text{PR}_{o_{\text{obj}}} = \alpha \text{PR}_{\text{InDeg}} + (1-\alpha) * \text{PR}_{w_p}, \tag{7}$$

where PR_{InDeg} is the PageRank value approximated by the in-degree value of vertex o_{obj} and PR_{w_p} is the total contribution that comes from all s_{sub} pointing to o_{obj}. $\alpha \in (0, 1)$. α is a hybrid factor that needs to be trained by some heuristic algorithm like simulated annealing. Using (4) and (5) we can get the following expression. With the expression, we can compute PageRank values more quickly:

$$\begin{aligned} \text{PR}_{o_{\text{obj}}} = {} & \alpha \left[\frac{d}{N} + \frac{1-d}{N} \frac{k_{\text{in}}}{\langle k_{\text{in}} \rangle}\right] + (1-\alpha) \\ & * \left[\frac{w_{p_i}}{\sum_{j: j \in s_k \to \forall p} w_{p_j}} \text{PR}\left(s_{\text{sub}}\right)\right]. \end{aligned} \tag{8}$$

3.2. Sliding Window Based Streaming Model. The original implementation of graph partitioner partitions graphs, which are stored in local files, into several parts. The vertices of graph G arrive in a stream with the set of edges. As vertices arrive, a partitioner decides to place the vertex on one computer of multiple machines. The partitioner can distribute the nodes randomly. The hash partitioning may lead to data skew and bad locality. If the partitioner can scan many nodes of the streaming graphs, it can process nodes in a batch and place nodes with closer relationship together. Inspired by this, we extend the model by implementing a buffer so that the partitioner may decide to place any node in the buffer, rather than the one at the front of the stream. The buffer is named as window (Figure 2). The window determines the boundary of the nodes that partitioner can process now. Once the nodes in the window are processed, the window slides by the window size. The sliding window ensures that partitioner processes nodes in a batch, instead of one node each time. Furthermore, in order to give the partitioner more flexibility, we allow it to access to the statistics of previous entire partitions.

3.3. Computing the Association between Vertices. For each vertex v in the sliding window, Algorithm 1 searches all start vertices in the sliding window which v can be reachable from, denoted by $S(v)$. And for each start vertex $u \in S(v)$, we compute the association value $A(u, v)$ for partitioning and denote the maximum start vertex set with $A(u, v)$ for vertex v as $AS(v) = \{u\}$. All the edges of a particular vertex are inserted into the sliding window at random order. Particularly, G_S means the subgraph of graph G made up by all vertices in

Input:
subgraph formed through sliding window: $G_S = (V_S, E_S)$
Output:
start vertex set for each vertex $\{S(v) \mid v \in V_S\}$,
Association set $\{AS(v) \mid v \in V_S\}$
(1) startVertexSet $\{sv_1, \ldots, sv_m\}$
(2) *Queue* $\leftarrow \{sv_1, \ldots, sv_m\}$
while Queue is not empty **do**
(3) $v \leftarrow$ Queue.front(); Queue.pop();
(4) **if** $v \in$ *startVertexSet* **do**
(5) $S(v) \leftarrow v$; $AS(v) \leftarrow v$;
(6) **foreach** $r \in e = (v, r)$
(7) **if** local indegree of $r > 0$ and local outdegree of $r > 0$
(8) Queue.push(r);
(9) **else**
(10) **foreach** $p \in S(u)$ $(u \in \text{edge}(u, v))$, $q \in AS(v)$ **do**
(11) **if** $A(p, v) > A(q, v)$ **do**
(12) **clean** $AS(v)$, $AS(v) \leftarrow \{p\}$;
(13) **else** $A(p, v) = A(q, v)$ **do**
(14) **update** $AS(v)$, $AS(v) \leftarrow \{p\}$;
(15) **if** $sv_{C_i} \in AS(u)$ and $sv_{C_j} \in AS(u)$ **do**
(16) $C_i' \leftarrow C_i \cup C_j$;

Algorithm 1: Retracing start vertex and computing association.

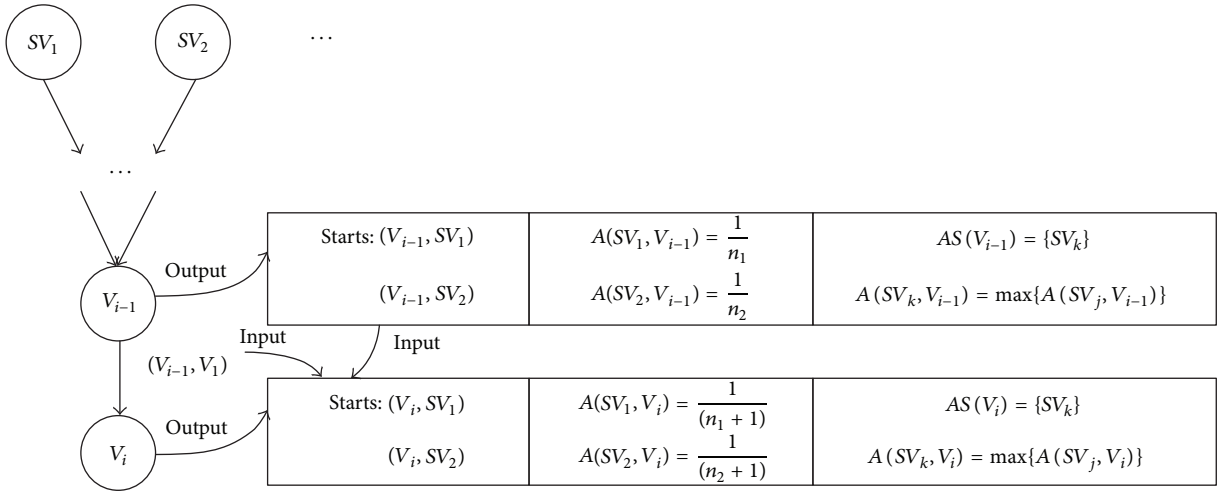


Figure 3: Search start vertices and computing association.

the sliding window. V_S means all vertices in G_S, E_S means all edges in G_S, and sv_{c_i} means the start vertex of cluster c_i.

This algorithm is implemented by iteratively traversing each vertex, which is similar to the breadth-first traversal from each start vertex. Specifically, in ith iteration, it processes the vertices which are an i-hop away from start vertices and gets the start vertex set from the predecessor of these vertices. It then computes $A(u, v)$ and updates AS according to start vertices. For example, in Figure 3, start vertex set of v_{i-1} is $\{sv_1, sv_2\}$, $A(sv_1, v_{i-1}) = 1/n_1$, and $A(sv_2, v_{i-1}) = 1/n_2$. Suppose $A(sv_1, v_{i-1}) = A(sv_2, v_{i-1})$; then $AS(v_{i-1}) = \{sv_1, sv_2\}$. Vertex v_{i-1} is the predecessor of vertex v_i, and by definition of *Association*, $A(sv_1, v_i) = 1/(n_1 + 1)$ and $A(sv_2, v_i) = 1/(n_2 + 1)$. The details are described in Algorithm 1.

3.4. AP-Based Streaming Graph Clustering. After the PageRank is computed, the vertices are divided into multiple subsets by their PageRank score, each of which contains the vertices that share similar PageRank scores. We use modified affinity propagation to cluster nodes. Affinity propagation [13] is an algorithm that takes input measures of similarity between pairs of data points and simultaneously considers all data points as potential exemplars. Unlike the clustering algorithms such as k-means or k-medoids, the number of clusters is not indicated for AP algorithm. AP algorithm also finds exemplars which represent the clusters contained in the input data, while k-medoids does the same thing. AP adopted a message passing algorithm which is called akin belief propagation. Messages are exchanged between data points until a high-quality set of exemplars and corresponding

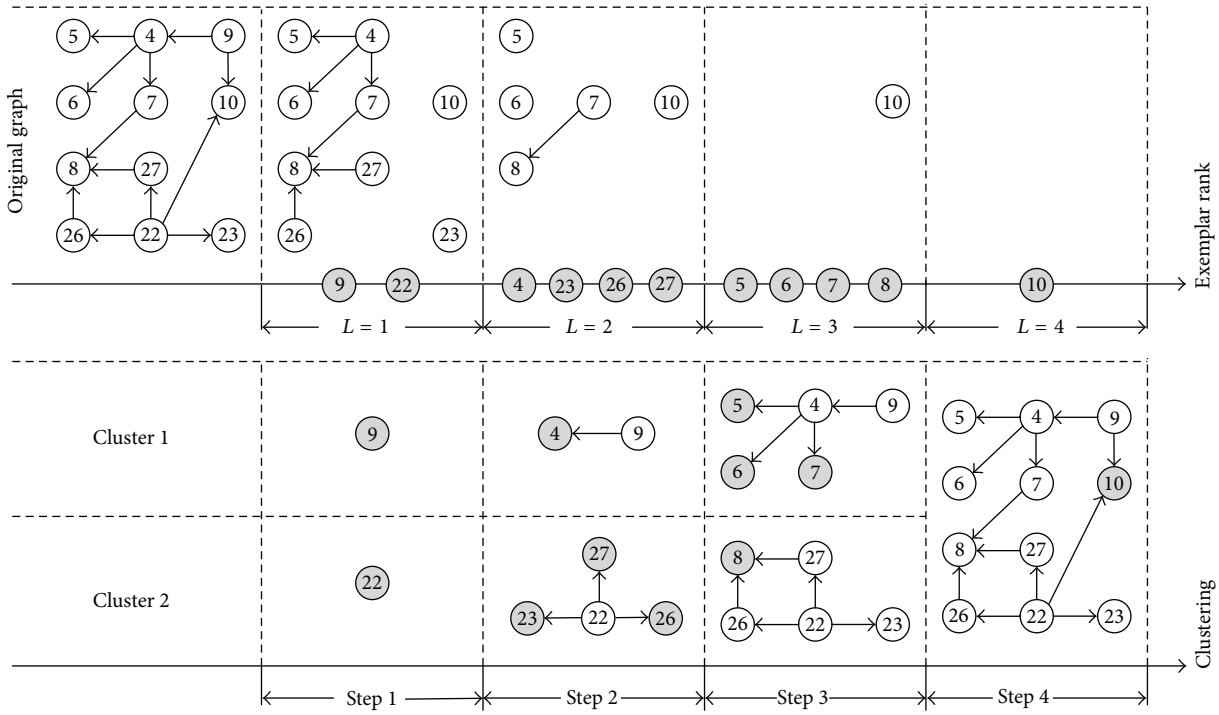


Figure 4: Association clustering.

clusters gradually emerges. If a maximal number of iterations are reached or the exemplars vary little within a fixed number of iterations, the algorithm will stop.

Here, the input data points for this algorithm are the PageRank scores of all the vertices in the sliding window. We use the Euclidean distance as the distance metric between data points, that is, the absolute value of their numerical difference. The affinity propagation algorithm then processes the data by alternating message between data points iteratively until a high-quality set of exemplars and corresponding clusters gradually emerges. Assume that the set of exemplars $\{ex_1, \ldots, ex_n\}$ is in a monotonically increasing order by PageRank score; that is, $PR(ex_1) < \cdots < PR(ex_m)$; then the L values for each exemplar are $L(ex_1) = 1 < \cdots < L(ex_n) = m$. The L values of data points in the corresponding cluster are the same as their exemplar. An example is given in Figure 4; $L(4) = L(23) = L(26) = L(27) = 2$ and $ex_2 = 4$. Algorithm 2 shows the association clustering implementation.

To reduce the overhead of memory and storage and speed up iterations, we simplify the association clustering into start vertices clustering. Note that the start vertices have a minimum L value in vertex ordering strategies; that is, $L(startvertex) = 1$. Based on this, a start vertex can stand for a cluster which contains all vertices at initial phase; for example, vertex 9 can stand for a cluster that merges all vertices with maximum *Association* from vertex 9, as shown in Figure 4.

Specifically, let R_n denote the clustering results at step n. Since a start vertex can stand for the corresponding cluster at initialization, obviously, R_0 is the clustering result at algorithm beginning and each start vertex is a cluster, respectively. The association clustering algorithm then iteratively groups the start vertices step by step. In the iteration at the nth step, for each pair of clusters in R_{n-1}, if the start vertices of these two clusters have the same value of *Association* with vertex u and $L(u) = n$, they will be merged into R_n. This continues until any two sets in R_n are disjoint. For example, in Figure 4, the initial clustering result is $R_0 = \{\{9\}, \{22\}\}$. Since $L(9) = L(22) = 1$, $AS(9) = \{9\}$, and $AS(22) = \{22\}$, then $R_1 = \{\{9\}, \{22\}\}$. Suppose that $L(4) = L(23) = L(26) = L(27) = 2$, $L(5) = L(6) = L(7) = L(8) = 3$, and $L(10) = 4$. At $step_1$, $R_2 = R_1 \cup \{AS(4), AS(23), AS(26), AS(27)\} = \{\{9\}, \{22\}, \{9\}, \{22\}, \{22\}, \{22\}\}$. We convert R_2 into a pairwise disjoint set; then $R_2 = \{\{9\}, \{22\}\}$. In similar way, $R_3 = \{\{9\}, \{22\}\}$. At $step_3$, $AS(10) = \{9, 22\}$, $R_4 = \{\{9\}, \{22\}, \{9, 22\}\}$. Finally, $R_4 = \{\{9, 22\}\}$.

As the iterations continue, the termination conditions are a potential problem. In order to generate the final results with acceptable redundancy, high intra-association, and well-balanced distribution, the strategy for the choice of termination conditions has to concentrate on two criteria: the number of clusters remaining (i.e., $|R_n|$ at step n) and the number of steps processed. Obviously, the more the steps are processed, the less the replicas and the higher the intra-associated partitioning results will be produced, but the fewer the number of clusters that will be remaining. Usually, we expect that the number of clusters remaining is larger than the number of storage nodes, for example, λ times compared with the number of storage nodes t.

After performing association clustering, we will assign each cluster to one partition, concentrating on the criterion which produces well-balanced data distribution. Meanwhile, the number of iterations should be as large as possible. In conclusion, the principle of determining the termination

Input:
subgraph formed through sliding window: $sG = (sV, sE)$
Output:
a k-way clustering result $\{P_1, \ldots, P_k\}$
(1) $R_0 \leftarrow \{\{sv_1\}, \{sv_2\}, \ldots, \{sv_m\}\}$, where $sv_i \in SV_{sG}$;
(2) $n \leftarrow 0$;
(3) **repeat**
(4) $n \leftarrow n + 1$;
(5) $R_n \leftarrow R_{n-1} \cup \{AS(u) \mid L(u) = n,\ u \in \{sV - sV_{\text{start}}\}\}$;
(6) **foreach** pair of $C_i, Clu_j \in R_n, C_i \neq C_j$ **do**
(7) **if** $sv_{C_i} \in AS(u)$ and $sv_{C_j} \in AS(u)$ **do**
(8) $C_i' \leftarrow C_i \cup C_j$;
(9) $R_n \leftarrow R_n \cup C_i'$;
(10) $R_n \leftarrow R_n - \{C_i\} - \{C_j\}$;
(11) **until** $|R_n| \leq \lambda k$ or $n > L_{\max}$;
(12) **foreach** $C_i \in R_{n-1}$ **do**
(13) $C_i \leftarrow \text{ExtendByAssociation}(C_i)$;
(14) **foreach** C_i **do**
(15) $P_j \leftarrow \text{DataDistributing}(C_i)$;

Algorithm 2: Association based clustering.

conditions is to merge as many clusters as possible on the basis of keeping the number of remaining clusters larger than times of the number of storage nodes. Hence, the problem can then be formulated as

$$\begin{aligned} \min \quad & \{n\} \\ \text{Subject to} \quad & \left|R_n\right| \leq \lambda t. \end{aligned} \tag{9}$$

Then, the iteration of association clustering will stop at step $n - 1$, if $|R_n| \leq \lambda t$ or n is larger than the maximal L value, where λ is a user-specified parameter for controlling the degree of data skew. For example, in Figure 4, supposing $t = 1$ and $\lambda = 1$, the iteration is stopped at step 2, in which there are two partitions remaining. After that, each final cluster will be generated by extending each start vertex in this cluster with all the vertices that have the best association with this start vertex. Finally, the function *DataDistributing* gives a novel strategy for selecting the partition to which the cluster is assigned. This strategy aims at a strong association and well-balanced distribution.

Association clustering generates clusters with different scales. The overhead of processing a small cluster is the same as that of a larger cluster. To avoid processing small clusters, we set a scale threshold for the clusters. A cluster that does not match this condition will be left behind in the sliding window and be processed when new data arrive. As each cluster may include one or more exemplars after association clustering, we want a unique exemplar to represent each cluster for data distributing. When one cluster has more than one exemplar, we select an exemplar from these exemplars through a secondary clustering: running AP again on these exemplars. If the result still has more than one exemplar, we choose the one with the maximum PageRank score. This can guarantee that the new generated exemplar has a relation with other vertices. Moreover, it prevents choosing a vertex with a large in-degree and results in skewed data placement.

Before the first bunch of data is processed, all storage nodes are empty. We use a greedy strategy for selecting the storage node to which cluster is assigned. This strategy aims at a well-balanced distribution. It sorts all clusters in decreasing order by the number of vertices they obtain. Then it assigns clusters one-by-one to minimal size partitions. It is notable that this strategy only processes little parts of the streaming graph and does not have effect over all distribution.

After initialization, the normal process of a cluster is to decide which storage node is to be distributed. First of all, we build an *ExIndex* for each storage node, which records each cluster's exemplar that has been already stored in the node in decreasing order by PageRank score, more specific information including the in-degree of the exemplar in order to approximate and update PageRank score. After association clustering, for each cluster, we send the PageRank of the exemplar PR_e to each storage node by multithread and compare the PageRank with the last one PR_i in the *ExIndex*. If $\text{PR}_e \geq \text{PR}_i$, we consider the cluster to be in association with data in this storage node and search the *ExIndex*. If this exemplar has already existed, we send a signal y back to the main node and otherwise n. According to all signals, the main node decides how to distribute the cluster. If more than one storage node sends y back, the cluster will be sent to the storage node with minimal storage capacity. If there is no signal y, then other clusters are processed first and the cluster is sent to the storage node with maximal storage space.

4. Experimental Evaluation

The experimental results are presented and analyzed in this section. Regarding the quality of partitions, we use *edge cut* (ec) size and *communication volume* (cv) with respect to optimizing graph query responding time. In addition, imbalance is also taken into consideration. We let imbalance be the fraction which equals the quotient between the size

Table 2: Statistics of real graphs.

Datasets	Amazon	EnWikt	DBLP	Yago
Vertices	735268	101355853	986285	2635316
Edges	5158014	4206756	6707236	5259414

Table 3: Edge cuts of real graphs.

Datasets	Amazon	EnWikt	DBLP	Yago
METIS	340163	30743970	1125114	1199683
HASH	5017956	95303805	6486516	4941314
DG	2254099	64543160	2918274	4674657
LDG	2204099	62750478	2663968	4239179
EDG	2252630	64543160	2958376	4614849
T	1292032	79535779	1292032	3388482
LT	1312032	79848943	1312032	3388662
ET	1310053	79624375	1310053	3388550
Assc	1113146	53410947	1154243	3254783

of the largest partition and the balanced (average) partition size.

We set random hash method as upper bound with the consideration that hash ignores both the structure of the graph and the locality among the edges completely. As a result, hash can achieve balanced partitions but incurs lots of edge-cutting across these partitions. We also set METIS as lower bound because it is an offline algorithm and can obtain the whole information of the graph to generate good partitions.

We also compare our solutions (short for Assc) to two typical kinds of partitioning algorithms: METIS (*gpmetis* with default configuration) as well as a collection of streaming partitioning algorithms including hash and methods proposed by Stanton and Kliot [14]. We also set the number of storage nodes k to a constant value equal to 16 and the value of parameter α to 0.5. We process unlabeled graphs with an edge weight as a constant value of 1.

All experiments were performed on a single machine which has a 16-core Intel Xeon CPU E7420 and 48 GB memory.

4.1. Real Graphs. We evaluate our approach over several real-world datasets which is generated from the real streaming case: *Amazon*, a symmetric graph describing the similarity among books; *English-Language Wiktionary*, a collaborative project to produce a free-content multilingual dictionary; *DBLP*, a bibliography service from which an undirected scientific collaboration network can be extracted; and *Yago*, a huge semantic knowledge base, derived from *Wikipedia*, *WordNet*, and *GeoNames*. Table 2 shows the basic information of the real graphs.

Tables 3 and 4 show the quality of the partitioning for the real-world graph. The quality is measured by ec and cv, respectively. The results were obtained by running the streaming partitioning methods and METIS. It is worth noting that we do not present communication volumes of other streaming graph partitioning methods in Table 4. As mentioned above, METIS is an offline algorithm that obtains the whole information of graph, whereas streaming graph partitioning methods can only pick up partial knowledge of the graph. It is reasonable that METIS surpasses streaming graph partitioning methods which can be proved by Table 3, so we omit cvs of other methods for succinctness.

Table 4: Communication volume situation of real graphs.

Datasets	Amazon	EnWikt	DBLP	Yago
METIS	407167	10021079	12227014	21026877
HASH	4316902	36827428	33183916	122343781
Assc	507354	14162380	15659341	15394621

Tables 3 and 4 give a close look at the edge-cutting number of each partitioning approach. It is clear that our solution produced partitions with significantly better quality reaching up to 1.17x (from 1.04x) over all other streaming partitioning methods, while METIS performs the best as we expected. The results in Table 3 also show that our method is comparable to METIS in terms of the number of edge cuts. We omit the experimental results of imbalance, for the reason that both our method and other competitive methods generated well-balanced partitions over all the graph datasets. It is worth noting that the maximal fraction represents imbalance equal to 0.04. So we ignore the results for briefness.

4.2. Synthetic Graphs. With the purpose of determining how effective our approach is in the context of graph query, we choose LUBM dataset with embedded communities to represent synthetic graphs. The LUBM dataset is widely used by the semantic web community for benchmarking triple stores. With the purpose of evaluating the scalability of our partitioning approach, we used six datasets of varying sizes generated by the LUBM data generator. Those LUBM datasets contain 10 M, 50 M, 100 M, 200 M, 300 M, and 500 M triples, respectively. We present the properties of these datasets in Table 5.

According to the experimental results, our solutions produced partitions with significantly better quality than HASH and METIS in terms of communication. Our method surpasses METIS at least 20% over all the LUBM datasets. For a straightforward view, we convert the data in these tables to Figure 5, which shows the communication traffic during the graph query with our system TripleBit [15, 16]. For the reason we explained above, we just present the results of METIS, HASH, and our methods.

Considering METIS holds the full knowledge of the graph, it is interesting that our method outperforms METIS just with fragmentary graph information acquired in streaming setting. It is worth noting that graph query is quite different from some typical graph algorithms like PageRank. It is the number of intrapartition edges and interpartition edges that dominates the execution time of PageRank. In this prerequisite, we can simply predict the performance of a graph algorithm with the quality of partitions split by one graph partitioner. However, graph query, unlike PageRank, is a subgraph patter matching problem. Its performance is more related to the distribution of subgraphs in each partition.

Table 5: Statistics of synthetic graphs.

Datasets	10 M	50 M	100 M	200 M	300 M	500 M
Vertices	3303724	16439317	32905170	65764621	98640459	164416780
Edges	13409395	66751196	133613894	267027610	400512826	667592614

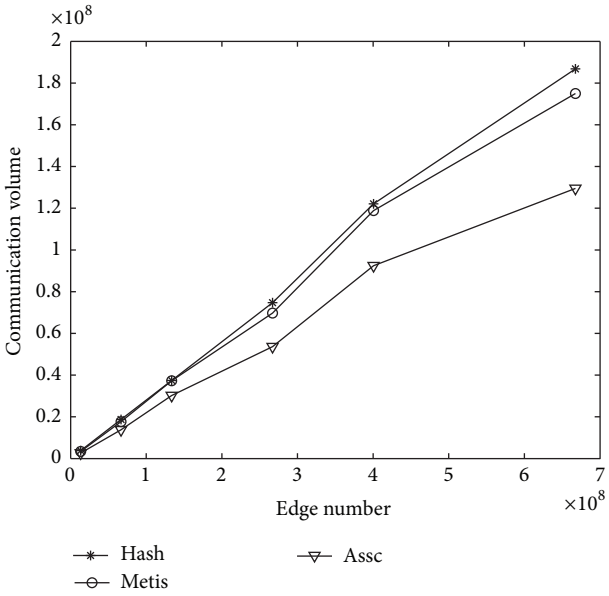


Figure 5: Communication during graph query.

Although METIS can lead to more balanced partitions with smaller fraction of edges cut, it ignores the subgraph associated with answers to graph query. Our method brings out larger number of edge cuts, admittedly. But by using *Association Based Clustering*, our method can preserve more subgraph pattern in each partition. In this way, when executing a graph query, a matched subgraph can be found within a partition boundary more likely. So compared to METIS, the communication traffic can be reduced by our method.

4.3. Discussion. METIS is based on a multilevel approach, which consists of three phases: the coarsening phase, the initial partitioning phase, and the uncoarsening phase. Among these three phases, coarsening phase plays a key role, which relies on the efficient choosing of objective functions, and for skewed graphs, many connected edges will be cut to avoid extremely large clusters. METIS can obtain the whole information of the graph structure and only the information of structure. As a result, it can generate good partitions with the cost of long execution time. As we explained above, in the scenario of graph query, it performs worse than our method.

Edge-based hash partitioning is a vertex-cut approach. This approach performs poorly when the graph structure is complex, which brings about the operations across partition boundaries. This consequently incurs frequent cross-node interactions coming with significant performance degeneration.

Our method considers the weight of edges by calculating PageRank score, which enables the vertices with high PageRank score to be merged late, giving a guarantee that we can get a reasonable size for the cluster. With association clustering, we can get high intra-associated clusters while the relations among vertices are well reserved, which means less *edge cut* and *communication volume*.

5. Related Work

We survey related work on static and streaming graph clustering from different angles. Regarding clustering problems, multiple algorithms have been proposed. K-means [17] is a simple and fast algorithm which often suffers from two major drawbacks. Firstly, the performance of K-means is highly related to the choice of the initial values which are treated as the cluster centers. Secondly, the objective function of the K-means is nonconvex coming with lots of local minima. K-means++ [18] algorithm was proposed which presents initialization procedure of cluster centers to overcome the initial centers sensitivity problem of a standard K-means. However, it also suffers from the local optimum problem. To add the ability for K-means++ to update its clusters as new points arrive from a data stream, StreamKM++ [19] was developed. StreamKM++ introduces the concept of a core set, and clusters are built on these core sets, which are good approximation of the clustering of the original points.

Being different from K-means, density-based clustering does not require the input of a predefined number of clusters and can form clusters with arbitrary shapes. One multiple stream density-based data processing algorithm is DenStream [20], which applies a damped window to DBSCAN [21]. DBSCAN is a popular density-based clustering algorithm with two simple parameters, which is effective, but is not designed to handle dynamic data streams. By extending the concept of core-objects to core-micro-clusters, DenStream [20] is able to cluster in the data stream environment.

Hash partitioning is one of the dominating approaches in graph partitioning. Lee and Liu present a novel semantic hash approach that utilizes access locality to partition big graphs across multiple computing nodes by maximizing the intrapartition processing capability and minimizing the interpartition communication cost [22]. Huang et al. use a graph partitioning algorithm instead of simple hash partitioning by source vertex, destination vertex, and labeled or unlabeled edge [23]. This allows vertices that are close together in the graph to be stored in the same machine. Stanton and Kliot proposed a weighted variant of the greedy algorithm for streaming balanced graph partitioning [14], which has an improvement over the hashing approach. Tsourakakis et al. presented a framework which unifies two seemingly orthogonal heuristics [24]. They also developed a streaming graph partitioning algorithm FENNEL based on this framework whose performance can be comparable to METIS.

Since each node in the graph has a natural PageRank property that can be used as a measure, regarding the computations performed over large graphs whose edges are presented in a streaming order, Sarma et al. proposed algorithms that require sublinear space and passes to compute the approximate PageRank values of vertices in a large graph [25]. Zhang et al. employed the affinity propagation algorithm and extend to data streaming to solve a challenge: how to cluster with the best representative [26]. Nguyen et al. present an algorithm that adopts eviction strategy to evaluate the likelihood of binding in terms of its contribution to a result in contrast to using a fixed time window for shedding the computation load [27]. Fischer et al. propose the use of graph partitioning algorithms to optimize the assignment of tasks to machines [28].

In order to process the growing data, many distributed infrastructures such as Hadoop [29] have been developed. Since Hadoop's batch-oriented synchronous nature does not satisfy the demand of real-time data processing, stream processing approaches based on information-flow processing have been proposed [28].

6. Conclusion and Future Work

In this paper, we propose a novel partitioning approach for streaming graph query in a general-purposed distributed storage system. Our approach, which is called association-oriented partitioning approach, adopts a hybrid approximate PageRank to retrace and compute associations between start vertices and other vertices. It then uses a hybrid PageRank-based affinity propagation clustering algorithm to generate several clusters, each including an exemplar. Finally, this method distributes different clusters with a brief strategy. This strategy judges whether the clusters have an association with the storage node. Extensive experiments conducted on these graphs prove that our method is effective. In the scenario of graph query, our method even outperforms METIS in terms of communication traffic across different partitions.

Our streaming graph partitioning approach needs to compute the rank values of vertices before distributing a vertex into a partition. The computation of PageRank value generally is a time-consuming step, although we have taken some strategies, including sliding window and approximate PageRank computation to improve the performance. Thus, our work will continue in two directions. *First*, we are working on extending our approach to distributed computing architecture. *Second*, we are interested in exploring the querying feature for labeled streaming graphs based our system.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

The paper was recommended to be published in journals by International Workshop on Big Data Programming and System Software 2016 (BDPSS 2016). The research is supported by Science and Technology Planning Project of Guangdong Province, China (nos. 2016B030306003 and 2016B030305002).

References

[1] "Linked open data project", 2014, http://linkeddata.org/.

[2] P. Wang, B. Xu, Y. Wu, and X. Zhou, "Link prediction in social networks: the state-of-the-art," *Science China Information Sciences*, vol. 58, no. 1, pp. 1–38, 2014.

[3] Z. Kaoudi, K. Kyzirakos, and M. Koubarakis, "Sparql query optimization on top of dhts," in *Proceedings of 9th International Semantic Web Conference (ISWC '10)*, pp. 418–435, Springer, 2010.

[4] A. Harth, J. Umbrich, A. Hogan, and S. Decker, "YARS2: a federated repository for querying graph structured data from the Web," *The Semantic Web*, vol. 4825, pp. 211–224, 2007.

[5] P. Yuan, W. Zhang, C. Xie, H. Jin, L. Liu, and K. Lee, "Fast iterative graph computation: a path centric approach," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, (SC '14)*, pp. 401–412, USA, November 2014.

[6] P. Yuan, C. Xie, L. Liu, and H. Jin, "Pathgraph: a path centric graph processing system," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 10, pp. 2998–3012, 2016.

[7] G. Malewicz, M. H. Austern, A. J. C. Bik et al., "Pregel: a system for large-scale graph processing," in *Proceedings of the International Conference on Management of Data (SIGMOD '10)*, pp. 135–146, ACM, June 2010.

[8] B. Wu, Y. Zhou, P. Yuan, L. Liu, and H. Jin, "Scalable SPARQL querying using path partitioning," in *Proceedings of the IEEE 31st International Conference on Data Engineering (ICDE '15)*, pp. 795–806, IEEE Computer Society, Seoul, South Korea, April 2015.

[9] "Metis", 2015, http://www.cs.umn.edu/metis/.

[10] "Lubm", 2014, http://swat.cse.lehigh.edu/projects/lubm/.

[11] S. Fortunato, M. Boguñá, A. Flammini, and F. Menczer, "Approximating pagerank from in-degree," in *Proceedings of 4th International Workshop on Algorithms and Models for the Web-Graph (WAW '06)*, vol. 4936, pp. 59–71, Springer, 2006.

[12] Y. Guo, Z. Pan, and J. Heflin, "LUBM: a benchmark for OWL knowledge base systems," *Web Semantics*, vol. 3, no. 2-3, pp. 158–182, 2005.

[13] B. J. Frey and D. Dueck, "Clustering by passing messages between data points," *American Association for the Advancement of Science. Science*, vol. 315, no. 5814, pp. 972–976, 2007.

[14] I. Stanton and G. Kliot, "Streaming graph partitioning for large distributed graphs," in *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2012*, pp. 1222–1230, chn, August 2012.

[15] P. Yuan, P. Liu, B. Wu, H. Jin, W. Zhang, and L. Liu, "Triplebit: a fast and compact system for large scale rdf data," *Proceedings of the VLDB Endowment*, vol. 6, no. 7, pp. 517–528, 2013.

[16] P. Yuan, C. Xie, H. Jin, L. Liu, G. Yang, and X. Shi, "Dynamic and fast processing of queries on large-scale rdf data," *Knowledge and Information Systems (KAIS)*, vol. 41, no. 2, pp. 311–334, 2014.

[17] G. Krishnasamy, A. J. Kulkarni, and R. Paramesran, "A hybrid approach for data clustering based on modified cohort intelligence and K-means," *Expert Systems with Applications*, vol. 41, no. 13, pp. 6009–6016, 2014.

[18] D. Arthur and S. Vassilvitskii, "k-means++: the advantages of careful seeding," in *Proceedings of the 18th Annual ACM-SIAM*

Symposium on Discrete Algorithms (SODA '07), pp. 1027–1035, Society for Industrial and Applied Mathematics (SIAM), 2007.

[19] M. R. Ackermann, Märtens M., C. Raupach, K. Swierkot, C. Lammersen, and C. Sohler, "StreamKM++: a clustering algorithm for data streams," *ACM Journal of Experimental Algorithmics*, vol. 17, Article 2.4, 30 pages, 2012.

[20] F. Cao, M. Ester, W. Qian, and A. Zhou, "Density-based clustering over an evolving data stream with noise," in *Proceedings of the 6th SIAM International Conference on Data Mining (SDM '06)*, pp. 328–339, SIAM, 2006.

[21] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the 2th International Conference on Knowledge Discovery and Data Mining (KDD '96)*, pp. 226–231, 1996.

[22] K. Lee and L. Liu, "Scaling queries over big rdf graphs with semantic hash partitioning," *Proceedings of the VLDB Endowment*, vol. 6, no. 14, pp. 1894–1905, 2013.

[23] J. Huang, D. J. Abadi, and K. Ren, "Scalable SPARQL querying of large RDF graphs," *Proceedings of the VLDB Endowment*, vol. 4, no. 11, pp. 1123–1134, 2011.

[24] C. Tsourakakis, C. Gkantsidis, B. Radunovic, and M. Vojnovic, "FENNEL: streaming graph partitioning for massive scale graphs," in *Prcedings of the 7th ACM International Conference on Web Search and Data Mining, WSDM 2014*, pp. 333–342, USA, February 2014.

[25] A. D. Sarma, S. Gollapudi, and R. Panigrahy, "Estimating PageRank on graph streams," *Journal of the ACM*, vol. 58, no. 3, Article ID 13, 2011.

[26] X. Zhang, C. Furtlehner, C. Germain-Renaud, and M. Sebag, "Data stream clustering with affinity propagation," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 7, pp. 1644–1656, 2014.

[27] M. K. Nguyen, T. Scharrenbach, and Bernstein A., "Seven commandments for benchmarking semantic flow processing systems," in *Proceedings of the 9th International Conference on Scalable Semantic Web Knowledge Base Systems (SSWS '13)*, pp. 66–80, Springer, 2013.

[28] L. Fischer, T. Scharrenbach, and A. Bernstein, "Scalable linked data stream processing via network-aware workload scheduling," in *Proceedings of the 9th International Conference on Scalable Semantic Web Knowledge Base Systems (SSWS '13)*, pp. 81–96, 2013.

[29] J. Shafer, S. Rixner, and A. L. Cox, "The hadoop distributed filesystem: balancing portability and performance," in *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS '10)*, pp. 122–133, usa, March 2010.